Documentation for Predicting Credit Card Default Using Development Data

1. Introduction

This document outlines the approach taken to predict the probability of credit card defaults using the development data provided. The dataset includes 96,806 records of credit card transactions with a flag, bad_flag, indicating whether a credit card has defaulted (i.e., bad_flag = 1 for defaults and bad_flag = 0 for non-defaults).

The objective is to develop a machine learning model that can predict the likelihood of default for unseen data (validation data). The final output will consist of two columns: the account_number (primary key) and the predicted probability of default for each credit card.

2. Approach

2.1 Data Exploration and Preprocessing

2.1.1 Data Cleaning

To handle missing data, we employed a tiered approach based on the percentage of missing values in each feature:

- Features with less than 5% missing values: For features with less than 5% missing values, we used mean imputation to fill the missing values. This method was chosen because the percentage of missing data is minimal, and imputing with the mean ensures the distribution of the feature remains consistent.
- Features with missing values between 5% and 30%: For features with missing values between 5% and 30%, we used K-Nearest Neighbors (KNN) imputation. KNN imputation works by finding the k-nearest neighbors for a data point and imputing the missing values based on the majority or average of those neighbors. This method helps in preserving the relationships between features and providing more context for imputation than the mean.
- Features with more than 30% missing values: Features with more than 30% missing data were removed from the dataset as they were deemed unreliable and likely to introduce noise. Retaining such features could lead to model instability and reduced performance.

By using this approach, we effectively handled missing data while ensuring the integrity of the dataset and minimizing any potential bias in model training.

2.1.2 Feature Engineering

- We identified features that might have high missing percentages and low correlation with the target variable (bad_flag). These features were removed to reduce noise in the model:
    - Missing Data: We calculated the percentage of missing values for each feature and removed those features with more than 30% missing values.
    - Correlation with Target: Features with a correlation lower than 0.1 with the target variable (bad_flag) were removed as they were likely to contribute little to the predictive power of the model.

2.1.3 Feature Scaling

In our approach, we did not perform any manual scaling of the features because XGBoost inherently handles unscaled features. This is due to the nature of the underlying algorithms XGBoost uses (i.e., decision trees), which are not sensitive to the scale of the input features.

However, XGBoost performs some inherent feature scaling and handling during the training process:

- Handling of Feature Values: XGBoost uses decision trees as its base learners, and decision trees inherently perform feature scaling through their splitting mechanism. During tree construction, XGBoost evaluates the best splits based on the feature values, without being influenced by the magnitude or range of those values. This allows the model to capture relationships in the data effectively without the need for feature normalization or standardization.
- Regularization: XGBoost applies both L1 and L2 regularization to the weights of the trees. This regularization process helps in controlling overfitting and is applied automatically to the model's feature weights, which can have an indirect scaling effect by shrinking less important features.
- Tree-based Feature Importance: XGBoost computes feature importance based on how often a feature is used in splits and how much it contributes to reducing the objective (loss function). This built-in mechanism helps in identifying important features and adjusting their influence in the model, without the need for explicit scaling of features.

Therefore, XGBoost's internal mechanisms allow it to handle unscaled features effectively, making manual feature scaling unnecessary for this model.

2.1.4 Target Variable Distribution

- The target variable (bad_flag) has an imbalanced distribution, with significantly more non-defaults than defaults. This class imbalance was addressed by using the scale_pos_weight parameter in the XGBoost model to give more weight to the minority class (defaults).

2.2 Model Selection and Training

2.2.1 Model Choice

In this project, we explored multiple classifiers to determine the best model for predicting the probability of credit card defaults. The models we considered included Logistic Regression, Random Forest, and Support Vector Machine (SVM). Each model was evaluated using cross-validation on the training set to assess its performance and select the most appropriate model for further tuning and testing.

Here are the training scores for each classifier based on 5-fold cross-validation:

- Logistic Regression: The model achieved a training score of 68.0% accuracy. Although it showed reasonable performance, the logistic regression model struggled with capturing the complexity of the data, especially due to the class imbalance.
- Random Forest: The Random Forest classifier performed better with a training score of 73.0% accuracy. This model's ability to handle complex relationships and its robustness against overfitting made it a strong candidate.
- SVM: The Support Vector Machine (SVM) model achieved a training score of 62.0% accuracy. Despite being a powerful model, SVM struggled to achieve higher performance, possibly due to the large feature set and the need for fine-tuning the hyperparameters.

Based on the cross-validation results, Random Forest demonstrated the best performance among the considered models and was ultimately selected for further tuning and evaluation. We found that XGBoost outperformed these models, so it became our final choice for model training. However, this comparison helped us understand the strengths and weaknesses of different approaches, and guided our decision to focus on XGBoost for its superior performance in handling class imbalance and its ability to capture complex patterns in the data.

```
Classifiers:  LogisticRegression Has a training score of 68.0 % accuracy score
Classifiers:  RandomForestClassifier Has a training score of 73.0 % accuracy score
Classifiers:  SVC Has a training score of 62.0 % accuracy score
```

2.2.2 Handling Class Imbalance

- In the development of the model, we initially applied SMOTE (Synthetic Minority Over-sampling Technique) to address the class imbalance between the two classes (bad_flag = 0 and bad_flag = 1). SMOTE works by generating synthetic samples for the minority class, which in this case is the class where bad_flag = 1 (the defaulted credit cards). The idea was to balance the dataset and improve model performance, particularly by preventing the model from being biased towards the majority class.
- However, after evaluating the model performance using SMOTE, we observed that the F1 score was poor, indicating that the synthetic data did not improve the model's ability to make accurate predictions, especially for the minority class. The key issue was that SMOTE, while it helped balance the class distribution, introduced noise into the dataset by generating synthetic examples that were not always representative of real-world data. This noise can sometimes mislead the model, leading to lower predictive performance.
- As a result, we decided to abandon SMOTE and focus on XGBoost's class weight adjustment mechanism, specifically using the scale_pos_weight parameter.
- scale_pos_weight = Negative Class Count/Positive Class Count
- This approach allowed the model to account for the class imbalance without the additional complexity and potential drawbacks of synthetic data generation. By calculating the scale of

positive and negative classes and adjusting the model's loss function accordingly, we were able to improve the model's balance between precision and recall, ultimately leading to a better F1 score.

- Thus, while SMOTE was initially considered to address class imbalance, it was found to be less effective than using class weight adjustments in XGBoost, which provided a more robust solution to our problem.

2.2.3 Model Training

- The model was trained using the training data (X_train and y_train) after handling missing data and feature selection. The trained model was then evaluated using the test set (X_test and y_test).

3. EVALUATION METRICS

3.1 Accuracy Score

- The model's overall accuracy was calculated to measure the percentage of correct predictions:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

3.2 Classification Report

- In addition to accuracy, we also evaluated the model using a classification report, which provides detailed metrics such as:
    - Precision: The percentage of correctly predicted defaults (True Positives) out of all predicted defaults.
    - Recall: The percentage of correctly predicted defaults out of all actual defaults.
    - F1-Score: The harmonic mean of Precision and Recall, providing a balance between the two.

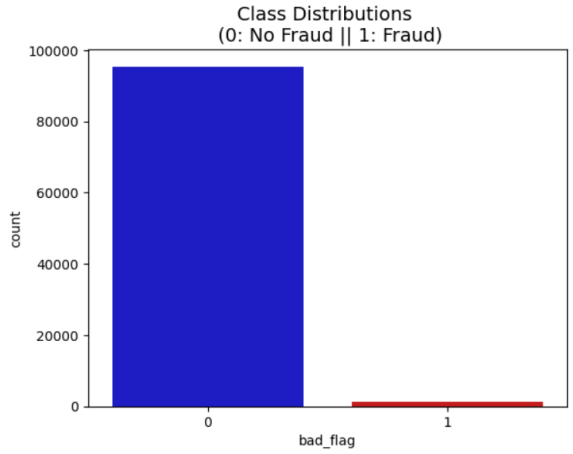
3.3 Precision-Recall Tradeoff

- Due to the class imbalance, we prioritized recall over precision to minimize false negatives (missed defaults). This is especially important because failing to predict a default can be much more costly for the bank than incorrectly predicting a default.

4. INSIGHTS AND OBSERVATIONS

4.1 Data Imbalance

- The data exhibited a clear class imbalance, with the majority of credit cards not defaulting (bad_flag = 0). This imbalance required special attention during model training. Without addressing this issue, the model could have easily learned to predict 0 (non-default) for all cases, resulting in a high accuracy of approximately 98.5%. However, this would be misleading, as the model would fail to predict the minority class (bad_flag = 1), which is the primary focus of the analysis.



- To mitigate this bias toward the majority class and ensure better predictive performance for both classes, we used XGBoost's scale_pos_weight parameter. This adjustment allowed the model to account for the class imbalance by applying a higher weight to the minority class. This prevented the model from simply predicting the majority class for all instances, resulting in a more balanced approach that improved the model's ability to identify and predict defaults (bad_flag = 1).
- By using class weight adjustments instead of relying on accuracy alone, we ensured that the model was not biased and could effectively predict both classes.

4.2 Feature Selection

- The feature selection process revealed that many features had either too much missing data or low correlation with the target variable. Removing these features helped improve model performance by reducing noise.

4.3 Model Performance

- The XGBoost model performed well on the test set, with good precision and recall scores for predicting defaults. However, improvements could be made by further tuning hyperparameters and testing other models.

5. CONCLUSION

This approach successfully tackled the problem of predicting credit card defaults using XGBoost. We handled missing data, addressed class imbalance, and selected relevant features. The model's performance was evaluated using accuracy and classification metrics, and the next steps involve further model refinement and hyperparameter optimization.

This document provides a comprehensive overview of the steps taken, from data preprocessing to model evaluation, along with insights into the data and potential improvements to the solution.